CHAPTER 9

Administering MongoDB

"Administering MongoDB is not like administering traditional RDBMS databases. Although most of the administrative tasks are not required or are done automatically by the system, still there are few tasks that need manual intervention."

In this chapter, you will go over the process of basic administrative operations for backups and restoration, importing and exporting data, managing the server, and monitoring the database instances.

Administration Tools

Before you dive into the administration tasks, here's a quick overview of the tools. Since MongoDB does not have a GUI-style administrative interface, most of the administrative tasks are done using the command line mongo shell. However, some UIs are available as separate community projects.

mongo

The mongo shell is part of the MongoDB distribution. It's an interactive JavaScript shell for the MongoDB database. It provides a powerful interface for administrators as well as developers to test queries and operations directly with the database.

In previous chapters, you covered development using the shell. In this chapter, you will go through the system administration tasks using the shell.

Third-Party Administration Tools

A number of third party tools are available for MongoDB. Most of the tools are web-based.

A list of all of the third party administration tools that support MongoDB is maintained by 10gen on the MongoDB web site at `https://docs.mongodb.org/ecosystem/tools/administration-interfaces/`.

S.G. Edward and N. Sabharwal, *Practical MongoDB*, DOI 10.1007/978-1-4842-0647-8_9

Backup and Recovery

Backup is one of the most important administrative tasks. It ensures that the data is safe and in case of any emergency can be restored back.

If the data cannot be restored back, the backup is useless. So, after taking a backup, the administrator needs to ensure that it's in a usable format and has captured the data in a consistent state.

The first skill an administrator needs to learn is how to take backups and restore it back.

Data File Backup

The easiest way to back up the database is to copy the data into the data directory folder.

All of the MongoDB data is stored in a data directory, which by default is `C:\data\db` (in Windows) or `/data/db` (in LINUX). The default path can be changed to a different directory using the `-dbpath` option when starting the mongod.

The data directory content is a complete picture of the data that is stored in the MongoDB database. Hence taking a MongoDB backup is simply copying the entire contents of the data directory folder.

Generally, it is not safe to copy the data directory content when MongoDB is running. One option is to shut down the MongoDB server before copying the data directory content.

If the server is shut down properly, the content of the data directory represents a safe snapshot of the MongoDB data, so it can be copied before the server is restarted again.

Although this is a safe and effective way of taking backups, it's not an ideal way, because it requires downtime.

Next, you will discuss techniques of taking backups that do not require downtime.

mongodump and mongorestore

mongodump is the MongoDB backup utility that is supplied as part of the MongoDB distribution. It works as a regular client by querying a MongoDB instance and writing all the read documents to the disk.

Let's perform a backup and then restore it to validate that the backup is in usable and consistent format.

The following code snippets are from running the utilities on a Windows platform. The MongoDB server is running on the localhost instance.

Open a terminal window and enter the following command:

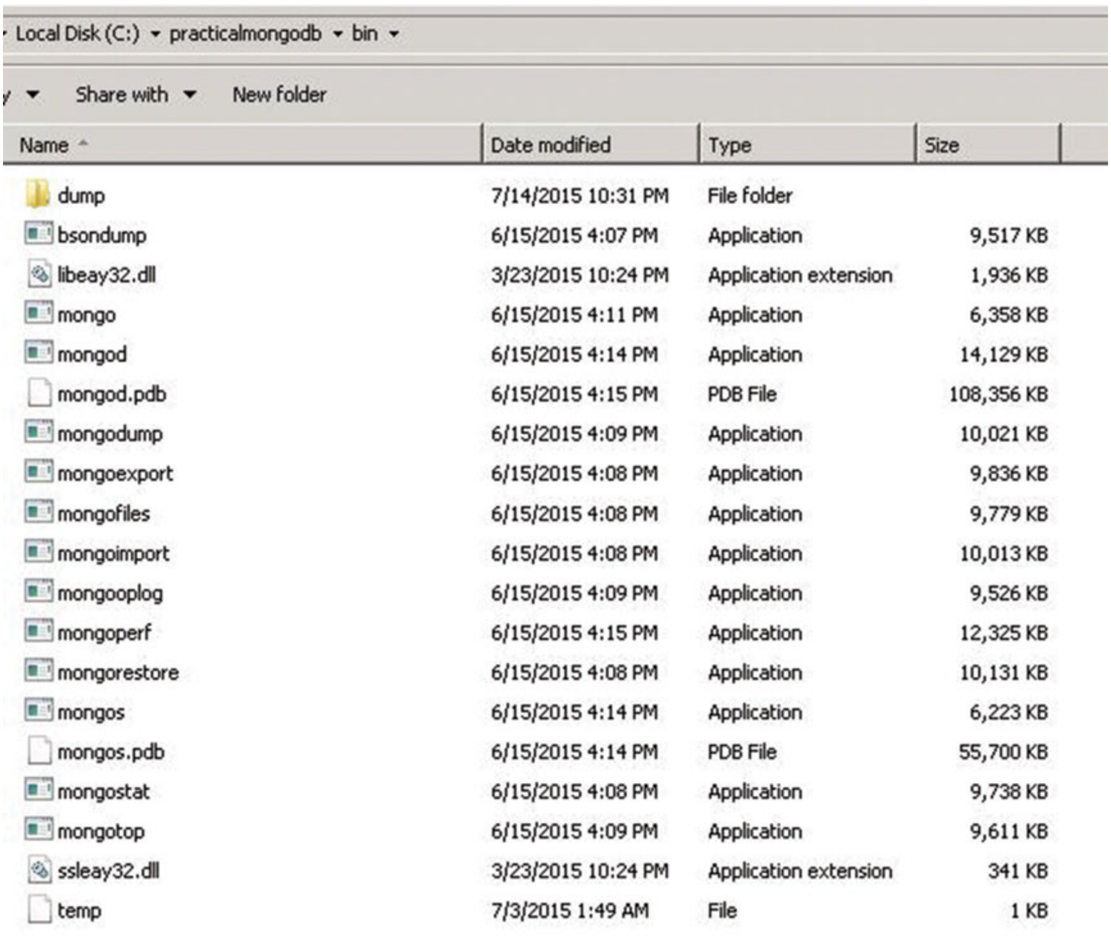
```
C:\>cd c:\practicalmongodb\bin
c:\practicalmongodb\bin>mongod --rest
2015-07-15T22:26:47.288-0700 I CONTROL  [initandlisten] MongoDB starting : pid=3820
port=27017 dbpath=c:\data\db\ 64-bit host=ANOC9
..........................................................................

2015-07-15T22:28:23.563-0700 I NETWORK  [websvr] admin web console waiting for connections
on port 28017
```

In order to run mongodump, execute the following in a new terminal window:

```
C:\>cd c:\practicalmongodb\bin
c:\practicalmongodb\bin>mongodump
2015-07-15T22:29:41.538-0700 writing admin.system.indexes to dump\admin\system.indexes.bson
................................
2015-07-14T22:29:46.720-0700    writing mydbproc.users to dump\mydbproc\users.bson
c:\practicalmongodb\bin>
```

This dumps the entire database under the dump folder in the bin folder directory itself, as shown in Figure 9-1.



Name	Date modified	Type	Size
dump	7/14/2015 10:31 PM	File folder	
bsondump	6/15/2015 4:07 PM	Application	9,517 KB
libeay32.dll	3/23/2015 10:24 PM	Application extension	1,936 KB
mongo	6/15/2015 4:11 PM	Application	6,358 KB
mongod	6/15/2015 4:14 PM	Application	14,129 KB
mongod.pdb	6/15/2015 4:15 PM	PDB File	108,356 KB
mongodump	6/15/2015 4:09 PM	Application	10,021 KB
mongoexport	6/15/2015 4:08 PM	Application	9,836 KB
mongofiles	6/15/2015 4:08 PM	Application	9,779 KB
mongoimport	6/15/2015 4:08 PM	Application	10,013 KB
mongooplog	6/15/2015 4:09 PM	Application	9,526 KB
mongoperf	6/15/2015 4:15 PM	Application	12,325 KB
mongorestore	6/15/2015 4:08 PM	Application	10,131 KB
mongos	6/15/2015 4:14 PM	Application	6,223 KB
mongos.pdb	6/15/2015 4:14 PM	PDB File	55,700 KB
mongostat	6/15/2015 4:08 PM	Application	9,738 KB
mongotop	6/15/2015 4:09 PM	Application	9,611 KB
ssleay32.dll	3/23/2015 10:24 PM	Application extension	341 KB
temp	7/3/2015 1:49 AM	File	1 KB

Figure 9-1. *The dump folder*

The mongodump utility by default connects to the localhost interface of the database on the default port.

Next, it pulls and stores each database and collection's associated data files into a predefined folder structure, which defaults to `./dump/[databasename]/[collectionname].bson`.

The data is saved in `.bson` format, which is similar to the format used by MongoDB for storing its data internally.

If content is already in the directory, it will remain untouched unless the dump contains same file. For example, if the dump contains the files `c1.bson` and `c2.bson`, and the output directory has files `c3.bson` and `c1.bson`, then mongodump will replace the `c1.bson` file of the folder with its `c1.bson` file, and will copy the `c2.bson` file, but it won't remove or change the `c3.bson` file.

You should make sure that the directory is empty before using it for mongodump unless you have a requirement of overlaying the data in your backups.

Single Database Backup

In the above example, you executed mongodump with the default setting, which dumps all of the databases on the MongoDB database server.

In a real-life scenario, you will have multiple application databases running on a single server, each having a different requirement of backup strategies.

Specifying the `-d` parameter in the mongodump utility will let you take the backup's database wise.

```
c:\practicalmongodb\bin>mongodump -d mydbpoc
2015-07-14T22:37:49.088-0700    writing mydbproc.mapreducecount1 to dump\mydbproc\
                                mapreducecount1.bson
......................
2015-07-14T22:37:54.217-0700    writing mydbproc.users metadata to dump\mydbproc\
                                users.metadata.json
2015-07-14T22:37:54.218-0700    done dumping mydbproc.users
c:\practicalmongodb\bin>
```

As of MongoDB-2.6, database administrator must have access to admin database in order to backup users and user-defined roles for given database as MongoDB stores these information in admin database only.

Collection Level Backup

There are two types of data in every database: data that changes rarely, such as configuration data where you maintain the users, their roles, and any application-related configurations, and then you have data that changes frequently such as the events data (in case of a monitoring application), posts data (in case of blog application), and so on.

As a result, the backup requirements are different. For instance, the complete database can be backed up once a week whereas the rapidly changing collection needs to be backed up every hour.

Specifying the `-c` parameter in the mongodump utility enables the user to implement backups for a specified collection individually.

```
c:\practicalmongodb\bin>mongodump -d mydbpoc -c users
2015-07-14T22:41:19.850-0700    writing mydbproc.users to dump\mydbproc\users.bson
2015-07-14T22:41:30.710-0700    writing mydbproc.users metadata to dump\mydbproc\
                                users.metadata.json
...........................................................
2015-07-14T22:41:30.712-0700    done dumping mydbproc.users
c:\practicalmongodb\bin>
```

If the folder where the data needs to be dumped is not specified, by default it dumps the data in a directory named `dump` in the current working directory, which in this case is `c:\practicalmongodb\bin`.

mongodump –Help

You have covered the basics of executing mongodump. Apart from the options mentioned above, mongodump provides other options that let you tailor the backups as per requirements. As with all other utilities, executing the utility with the `-help` option will provide the list of all available options.

mongorestore

As mentioned, it is mandatory for the administrators to ensure that the backups are happening in a consistent and usable format. So the next step is to restore the data dump back using mongorestore.

This utility will restore the database back to the state when the dump was taken. Prior to version 3.0, it was allowed to run the command without even starting the mongod/mongos. Starting from version 3.0, if the command is executed before starting the mongod/mongos the following error(s) will show:

```
c:\>cd c:\practicalmongodb\bin
c:\ practicalmongodb\bin>mongorestore

2015-07-15T22:43:07.365-0700    using default 'dump' directory
2015-07-15T22:43:17.545-0700    Failed: error connecting to db server: no reachable servers
```

You must run the mongod/mongos instance prior to running the `mongorestore` command.

```
c:\>cd c:\practicalmongodb\bin
c:\ practicalmongodb\bin>mongod --rest
2015-07-15T22:43:25.765-0700 I CONTROL  [initandlisten] MongoDB starting : pid=3820
                                        port=27017 dbpath=c:\data\db\ 64-bit host=ANOC9
.............................................................................
2015-07-15T22:43:25.865-0700 I NETWORK  [websvr] admin web console waiting for connections
                                        on port 28017
c:\ practicalmongodb\bin>mongorestore
2015-07-15T22:44:09.786-0700    using default 'dump' directory
2015-07-15T22:44:09.792-0700    building a list of dbs and collections to restore from dump dir
...................................
2015-07-15T22:44:09.732-0700    restoring indexes for collection mydbproc.users from metadata
2015-07-15T22:44:09.864-0700    finished restoring mydbproc.users
c:\practicalmongodb\bin>
```

This force appends the data to the back of the existing data.

To override the default behavior, `-drop` should be used in the above snippet.

The `-drop` command indicates to the mongorestore utility that it needs to delete all the collections and data within the aforementioned database and then restore the dump data back to the database.

If `-drop` is not used, the command appends the data to the end of the existing data.

Note that starting from version 3.0, the `mongorestore` command can also accept input from standard input.

Restoring a Single Database

As you saw in the backup section, the backup strategies can be specified at individual database level. You can run mongodump to take a backup of a single database by using the -d option.

Similarly, you can specify the -d option to mongorestore to restore individual databases.

```
c:\ practicalmongodb\bin>mongorestore  -d mydbpocc:\practicalmongodb\bin\dump\mydbproc  -drop
2015-07-14T22:47:01.155-0700    building a list of collections to restore from
C :\practicalmongodb\bin\dump\mydbproc dir
2015-07-14T22:47:01.156-0700 reading metadata file from
C :\practicalmongodb\bin\dump\mydbproc \users.metadata.json
..........................................................................
2015-07-14T22:50:09.732-0700    restoring indexes for collection mydbproc.users from metadata
2015-07-14T22:50:09.864-0700    finished restoring mydbproc.users
c:\practicalmongodb\bin>
```

Restoring a Single Collection

As with mongodump where you can use -c option to specify collection-level backups, you can also restore individual collections by using the -c option with the mongorestore utility.

```
c:\ practicalmongodb\bin>mongorestore -d mydbpoc -c users
C:\ practicalmongodb\bin\dum\mydb\user.bson -drop

2015-07-14T22:52:14.732-0700    restoring indexes for collection mydbproc.users from metadata
2015-07-14T22:52:14.864-0700    finished restoring mydbproc.users
c:\practicalmongodb\bin>
```

Mongorestore –Help

The mongorestore also has multiple options, which can be viewed using the -help option. Consult the following web site also: http://docs.mongodb.org/manual/core/backups/.

fsync and Lock

Although the above two methods (mongodump and mongorestore) enable you take a database backup without any downtime, they don't provide the ability to get a point-in-time data view.

You saw how to copy the data files to take the backups, but this requires shutting down the server before copying the data, which is not feasible in a production environment.

MongoDB's fsync command lets you copy content of the data directory by running MongoDB without changing any data.

The fsync command forces all pending writes to be flushed to the disk. Optionally, it holds a lock in order to prevent further writes until the server is unlocked. This lock only makes the fsync command usable for backups.

To run the command from the shell, connect to the mongo console in a new terminal window.

```
c:\practicalmongodb\bin>mongo
MongoDB shell version: 3.0.4
connecting to: test
>
```

Next, switch to admin and issue the runCommand to fsync:

```
>use admin
switched to db admin
>db.runCommand({"fsync":1, "lock":1})
{
        "info" : "now locked against writes, use db.fsyncUnlock() to unlock",
        "seeAlso" : "http://dochub.mongodb.org/core/fsynccommand",
        "ok" : 1
}
>
```

At this point, the server is locked for any writes, ensuring that the data directory is representing a consistent, point-in-time snapshot of the data. The data directory contents can be safely copied to be used as the database backup.

You must unlock the database post the completion of the backup activity. In order to do so, issue the following command:

```
>db.$cmd.sys.unlock.findOne()
{ "ok" : 1, "info" : "unlock completed" }
>
```

The currentOp command can be used to check whether the database lock has been released or not.

```
>db.currentOp()
{ "inprog" : [ ] }
 (It may take a moment after the unlock is first requested.)
```

The fsync command lets you take a backup without downtime and without sacrificing the backup's point-in-time nature. However, there is a momentary blocking of the writes (also called a momentary write downtime).

Starting from version 3.0, when using WiredTiger, fsync cannot guarantee that the data files will not change. So it cannot be used to ensure consistency for creating backups.

Next, you'll learn about slave backup. This is the only backup technique that enables taking a point-in-time snapshot without any kind of downtime.

Slave Backups

Slave backups are the recommended way for data backups in MongoDB. The slave always stores a data copy that is nearly in sync with the master, and the slave availability or performance is not much of an issue. You can apply any of the techniques discussed earlier on the slave rather than the master: shutting down, fsync with lock, or dump and restore.

Importing and Exporting

When you are trying to migrate your application from one environment to another, you often need to import data or export data.

mongoimport

MongoDB provides the mongoimport utility that lets you bulk load data directly into a collection of the database. It reads from a file and bulk loads the data into a collection.

These methods are not suitable for production environment.

The following three file formats are supported by mongoimport:

- JSON: In this format you have JSON blocks per line, which represent a document.
- CSV: This is a comma-separated file.
- TSV: TSV files are same as CSV files; the only difference is it uses a tab as the separator.

Using `-help` with `mongoimport` will provide all the options available with the utility. mongoimport is very simple. Most of the time you will end up using the following options:

- `-h` **or** `-host`**:** This specifies the mongod hostname where the data need to be restored. If the option is not specified, the command will connect to the mongod running on localhost at port 27017 by default. Optionally, a port number can be specified to connect to mongod running on a different port.
- `-d` **or** `-db`**:** Specifies the database where the data needs to be imported.
- `-c` **or** `-collection`**:** Specifies the collection where data need to be uploaded.
- `--type`**:** This is the file type (i.e. CSV, TSV or JSON).
- `--file`**:** This is the file path from where the data need to be imported.
- `--drop`**:** If this option is set, it will drop the collection and recreate the collection from the imported data. Otherwise, the data is appended at the end of the collection.
- `--headerLine`**:** This is used for CSV or TSV files only, and is used to indicate that the first line is a header line.

The following command imports the data from a CSV file to the `testimport` collection on the localhost:

```
c:\practicalmongodb\bin>mongoimport --host localhost --db mydbpoc --collection testimport
--type csv –file c:\exporteg.csv --headerline
2015-07-14T22:54:08.407-0700    connected to: localhost
2015-07-14T22:54:08.483-0700    imported 15 documents
c:\ practicalmongodb\bin>
```

mongoexport

Similar to the mongoimport utility, MongoDB provides a mongoexport utility that lets you export data from the MongoDB database. As the name suggests, this utility exports files from the existing MongoDB collections.

Using -help shows available options with the mongoexport utility. The following options are the ones you will end up using most:

- **-q:** This is used to specify the query that will return as output the records that need to be exported. This is similar to what you specify in the db.CollectionName.find() function when you have to retrieve records matching the selection criteria. If no query is specified, all the documents are exported.
- **-f:** This is used to specify the fields that you need to export from the selected documents.

The following command exports the data from Users collection to a CSV file:

```
c:\practicalmongodb\bin>mongoexport -d mydbpoc -c myusers -f _id,Age -type=csv > myusers.csv
2015-07-14T22:54:48.604-0700 connected to: 127.0.0.1
2015-07-14T22:54:48.604-0700 exported 22 records
c:\practicalmongodb\bin>
```

Managing the Server

In this section, you will look at the various options that you need to be aware of as an administrator of the system.

Starting a Server

This section covers how to start the server. Previously, you used the mongo shell to start the server by running mongod.exe.

The MongoDB server can be started manually by opening a command prompt (run as administrator) in Windows or a terminal window on Linux systems and typing the following command:

```
C:\>cd c:\practicalmongodb\bin
c:\ practicalmongodb\bin>mongod
mongod --help for help and startup options
.........................................
```

This window will display all the connections that are being made to the mongod. It also displays information that can be used to monitor the server.

If no configuration is specified, MongoDB starts up with the default database path of C:\data\db on Windows and /data/db on Linux and binds to the localhost using default ports 27017 and 27018.

Typing ^C will shut down the server cleanly.

MongoDB provides two methods for specifying configuration parameters for starting up the server.

The first is to specify using command-line options (refer to Chapter tk).

The second method is to load a configuration file. The server configuration can be changed by editing the file and then restarting the server.

Stopping a Server

The server can be shut down pressing CTRL+C in the mongod console itself. Otherwise, you can use the `shutdownServer` command from the mongo console.

Open a terminal window, and connect to the mongo console.

```
C:\>cd c:\practicalmongodb\bin
c:\practicalmongodb\bin>mongo
MongoDB shell version: 3.0.4
connecting to: test
>
```

Switch to admin db and issue the `shutdownServer` command:

```
>use admin
switched to db admin
>db.shutdownServer()
2015-07-14T22:57:20.413-0700 I NETWORK DBClientCursor::init call() failed server should be down...
2015-07-14T22:57:20.418-0700 I NETWORK trying reconnect to 127.0.0.1:27017
2015-07-14T22:57:21.413-0700 I NETWORK 127.0.0.1:27017 failed couldn't connect to server 127.0.0.1:27017
>
```

If you check the mongod console where you started the server in the previous step, you will see that the server has been shut down successfully.

```
.......................
2015-07-14T22:57:30.259-0700 I COMMAND  [conn1] terminating, shutdown command received
2015-07-14T22:57:30.260-0700 I CONTROL  [conn1] now exiting
.................................................
2015-07-14T22:57:30.380-0700 I STORAGE  [conn1] shutdown: removing fs lock...
2015-07-14T22:57:30.380-0700 I CONTROL  [conn1] dbexit:  rc: 0
```

Viewing Log Files

By default the entire log output of MongoDB is written to `stdout` but this can be changed by specifying the logpath option in the configuration when starting the server to redirect the output to a file.

The log file contents can be used to identify problems such as exceptions, which may indicate some data problem or connection issues.

Server Status

`db.ServerStatus()` is a simple method provided by MongoDB for checking the server status, such as number of connections, uptime, and so on. The output of the server status command depends upon the operating system platform, MongoDB version, storage engine used, and type of configuration (like standalone, replica set, and sharded cluster).

Starting from version 3.0, the following sections are removed from the output: workingSet, indexCounters, and recordStats.

In order to check the status of a server using the MMAPv1 storage engine, connect to the mongo console, switch to admin db, and issue the `db.serverStatus()` command.

```
c:\practicalmongodb\bin>mongo
MongoDB shell version: 3.0.4
connecting to: test
>use admin
switched to db admin
>db.serverStatus()
host" : "ANOC9",
 "version" : "3.0.4",
 "process" : "mongod",
 "pid" : NumberLong(1748),
 "uptime" : 14,
 "uptimeMillis" : NumberLong(14395),
"uptimeEstimate" : 13,
 "localTime" : ISODate("2015-07-14T22:58:44.532Z"),
"asserts" : {
"regular" : 0,
        "warning" : 0,
        "msg" : 0,
        "user" : 1,
        "rollovers" : 0
},
.........................................................
```

The above `serverStatus` output will also have a “backgroundflushing” section, which displays reports corresponding to the process used by MongoDB to flush data to disk using MMAPv1 as the storage engine.

The "opcounters" and "asserts" sections provide useful information that can be analyzed to classify any problem.

The “opcounters” section shows the number of operations of each type. In order to find out if there’s any problem, you should have a baseline of these operations. If the counters start deviating from the baseline, this indicates a problem and will require taking action to bring it back to the normal state.

The “asserts” section depicts the number of client and server warnings or exceptions that have occurred. If you find a rise in such exceptions and warnings, you need to take a good look at the logfiles to identify if a problem is developing. A rise in the number of asserts may also indicate a problem with the data, and in such scenarios MongoDB validate functions should be used to check that the data is undamaged.

Next, let’s start the server using the WiredTiger storage engine and see the serverStatus output.

```
c:\practicalmongodb\bin>mongod -storageEngine wiredTiger
2015-07-14T22:51:05.965-0700 I CONTROL  Hotfix KB2731284 or later update is installed, no
need to zero-out data files
2015-07-29T22:51:05.965-0700 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=1G,session_max=20000,eviction=(threads_max=4),statistics=(fast),log=(enabl
ed=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),
checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0)
..................................................
```

In order to check the server status, connect to the mongo console, switch to admin db, and issue the `db.serverStatus()` command.

```
c:\practicalmongodb\bin>mongo
MongoDB shell version: 3.0.4
connecting to: test
>use admin
switched to db admin
>db.serverStatus()

"wiredTiger" : {
"uri" : "statistics:",
"LSM" : {
"...........................................................,
"tree maintenance operations scheduled":0,
............................................................,
},
 "async" : {
         "number of allocation state races":0,
         "number of operation slots viewed for allocation":0,
         "current work queue length" : 0,
         "number of flush calls" : 0,
         "number of times operation allocation failed":0,
         "maximum work queue length" : 0,
............................................................,
 },
 "block-manager" : {
         "mapped bytes read" : 0,
         "bytes read" : 966656,
         "bytes written" : 253952,
           ...................................,
         "blocks written" : 45
 },
............................................................,
```

As you can see, the server status output has a new section known as wiredTiger statistics when started with storage engine WiredTiger.

Identifying and Repairing MongoDB

In this section, you will look at how you can repair a corrupt database.

If you are getting errors like

- Database server refuses to start, stating data files are corrupt
- Asserts are seen in the log files or `db.serverStatus()` command
- Strange or unexpected queries results

this means the database is corrupt and a repair must be run in order to recover the database.

The first thing you need to do before you can start the repair is to take the server offline if it's not already. You can use either option mentioned above. In this example, type `^C` in the mongod console. This will shut down the server.

Next, start the mongod using the -repair option, as shown:

```
c:\practicalmongodb\bin>mongod --repair
2015-07-14T22:58:31.171-0700 I CONTROL  Hotfix KB2731284 or later update is installed,
                                        no need to zero-out data files
2015-07-14T22:58:31.173-0700 I CONTROL  [initandlisten] MongoDB starting : pid=3996
                                        port=27017 dbpath=c:\data\db\ 64-bit host=ANOC9
2015-07-14T22:58:31.174-0700 I CONTROL  [initandlisten] db version v3.0.4
.....................................
2015-07-14T22:58:31.447-0700 I STORAGE  [initandlisten] shutdown: removing fs lock...
2015-07-14T22:58:31.449-0700 I CONTROL  [initandlisten] dbexit:  rc: 0
c:\ practicalmongodb\bin>
```

This will repair mongod. If you look at the output, you'll find various discrepancies that the utility is repairing. Once the repair process is over, it exits.

After completion of the repair process, the server can be started as normal and then the latest database backups can be used to restore missing data.

At times, you may notice that the drive is running out of disk space when a large database is under repair. This is due to the fact that the MongoDB needs to create a temporary copy of the files on the same drive as the data files. To overcome this issue, while repairing a database you should use the -repairpath parameter to specify the drive where the temporary files can be created during the repair process.

Identifying and Repairing Collection Level Data

Sometimes you might want to validate that the collection holds valid data and had valid indexes. For such cases, MongoDB provides a validate() method that validates the content of the specified collection.

The following example validates the data of the Users collection:

```
c:\practicalmongodb\bin>mongo
MongoDB shell version: 3.0.4
connecting to: test
>use mydbpoc
switched to db mydbpoc
>db.myusers.validate()
{
        "ns" : "mydbpoc.myusers",
        "firstExtent" : "1:4322000 ns:mydbpoc.myusers",
        "lastExtent" : "1:4322000 ns:mydbpoc.myusers",
        "...............
        "valid" : true,
        "errors" : [ ],
        "warning" : "Some checks omitted for speed. use {full:true} option to do
 more thorough scan.",
        "ok" : 1
}
```

Both the data files and the associated indexes are checked by default by the validate() option. The collection statistics are provided to help in identifying if there's any problem with the data files or the indexes.

If running `validate()` indicates that the indexes are damaged, in that case `reIndex` can be used to re-index the indexes of the collection. This drops and rebuilds all the indexes of the collection.

The following command reindexes the `Users` collection's indexes:

```
>use mydbpoc
switched to db mydbpoc
>db.myusers.reIndex()
{
        "nIndexesWas" : 1,
        "msg" : "indexes dropped for collection",
        "nIndexes" : 1,
        "indexes" : [
                {
                        "key" : {
                                "_id" : 1
                        },
                        "ns" : "mydbpoc.myusers",
                        "name" : "_id_"
                }
        ],
        "ok" : 1
}
>
```

If the collection's data files are corrupt, then running the `-repair` option is the best way to repair all of the data files.

Monitoring MongoDB

As a MongoDB server administrator, it's important to monitor the system's performance and health. In this section, you will learn ways of monitoring the system.

mongostat

mongostat comes as part of the MongoDB distribution. This tool provides simple stats on the server; although it's not extensive, it provides a good overview. The following shows the statistics of the localhost. Open a terminal window and execute the following:

```
c:\>cd c:\practicalmongodb\bin
c:\practicalmongodb\bin>mongostat
```

The first six columns show the rate at which various operations are handled by the mongod server. Apart from these columns, the following column is also worth mentioning and can be of use when diagnosing problems:

> Conn: This is an indicator of the number of connections to the mongod instance. A high value here can indicate a possibility that the connections are not getting released or closed from the application, which means that although the application is issuing an open connection, it's not closing the connection after completion of the operation.

Starting from version 3.0, mongostat can also return its response in json format using option -json.

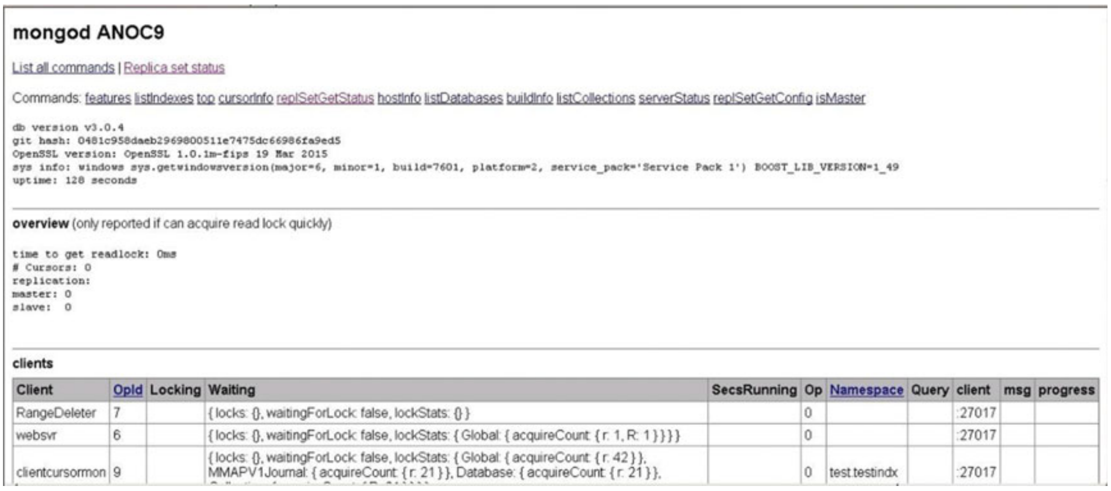
```
c:\>cd c:\practicalmongodb\bin
c:\practicalmongodb\bin>mongostat -json

{"ANOC9":{"ar|aw":"0|0","command":"1|0","conn":"1","delete":"*0","faults":"1","flushes":"0",
"getmore":"0","host":"ANOC9","insert":"*0","locked":"" ,"mapped":"560.0M","netIn":"79b",
"netOut":"10k","non mapped":"","qr|qw":"0|0","query":"*0","res":"153.0M","time":"05:16:17",
"update":"*0","vsize":"1.2G"}}
```

mongod Web Interface

Whenever a mongod is started up, it creates a web port by default, which is 1000 higher than the port number mongod uses to listen for a connection. By default the HTTP port is 28017.

This mongod web interface is accessed via your web browser, and it displays most of the statistical information. If the mongod is running on localhost and is listening for the connections on port 27017, then the HTTP status page can be accessed using the following URL: http://localhost:28017. The page looks like Figure 9-2.



mongod ANOC9

List all commands | Replica set status

Commands: features listIndexes top cursorInfo replSetGetStatus hostInfo listDatabases buildInfo listCollections serverStatus replSetGetConfig isMaster

```
db version v3.0.4
git hash: 0481c958daeb2969800511e7475dc66986fa9ed5
OpenSSL version: OpenSSL 1.0.1m-fips 19 Mar 2015
sys info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
uptime: 128 seconds
```

overview (only reported if can acquire read lock quickly)

```
time to get readlock: 0ms
# Cursors: 0
replication:
master: 0
slave:  0
```

clients

Client	OpId	Locking	Waiting	SecsRunning	Op	Namespace	Query	client	msg	progress
RangeDeleter	7		{ locks: {}, waitingForLock: false, lockStats: {} }		0			:27017		
websvr	6		{ locks: {}, waitingForLock: false, lockStats: { Global: { acquireCount: { r: 1, R: 1 } } } }		0			:27017		
clientcursormon	9		{ locks: {}, waitingForLock: false, lockStats: { Global: { acquireCount: { r: 42 } }, MMAPV1Journal: { acquireCount: { r: 21 } }, Database: { acquireCount: { r: 21 } },		0	test.testindx		:27017		

Figure 9-2. Web interface

Third-Party Plug-Ins

In addition to this tool, there are various third-party adapters available for MongoDB which let you use common open source or commercial monitoring systems such as cacti, Ganglia, etc. On its website, 10gen maintains a page that shares the latest information about available MongoDB monitoring interfaces.

To get an up-to-date list of third-party plug-ins, go to www.mongodb.org/display/DOCS/Monitoring+and+Diagnostics.

MongoDB Cloud Manager

In addition to the tools and techniques discussed above for monitoring and backup purposes, there is the MongoDB Cloud Manager (formerly known as MMS – MongoDB Monitoring Services). It's developed by the team who developed MongoDB and is free to use (30-day trial license). In contrast to the techniques discussed above, MongoDB Cloud Manager provides user interface as well as logs and performance details in the form of graphs and charts.

MongoDB Cloud Manager charts are interactive, enabling the user to set a custom date range, as depicted in Figure 9-3.

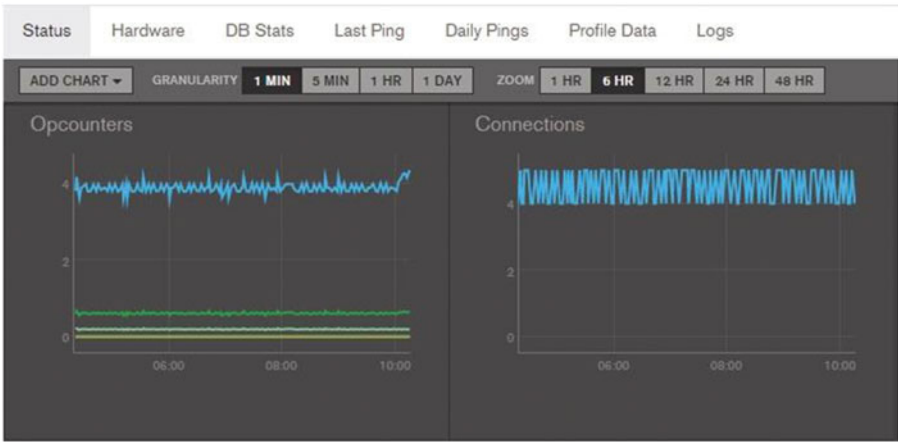


Figure 9-3. *Setting a custom date range*

Another neat feature of the Cloud Manager is the ability to use email and text alerts in case of different events. This is depicted in Figure 9-4.

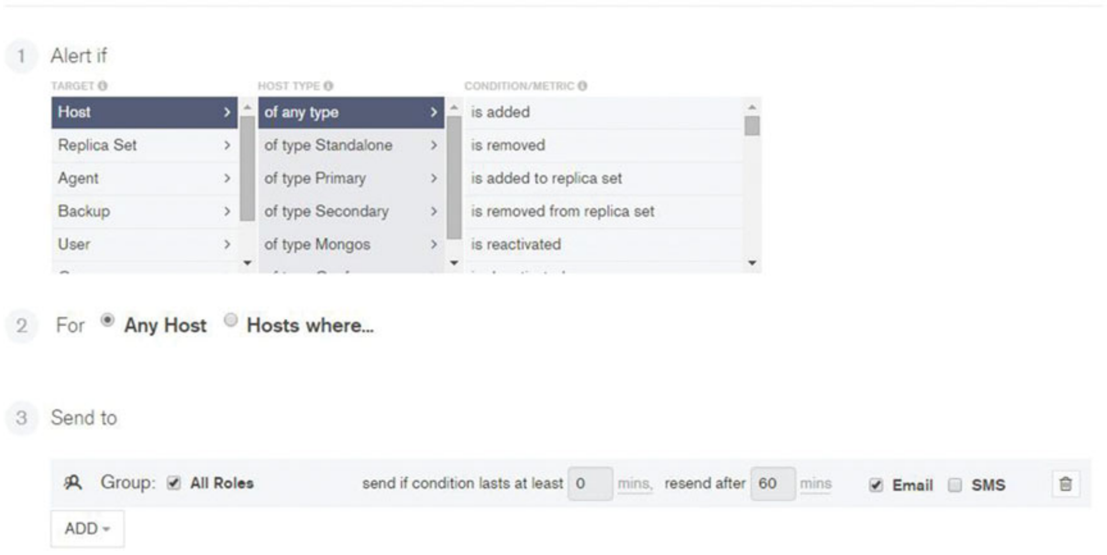


Figure 9-4. *Email and text alerts*

Not only does Cloud Manager provides graphs and alerts, it also lets you view the slower queries ordered by response time. You can easily see how your queries are performing all at one place. Figure 9-5 shows the graph that charts query performance.

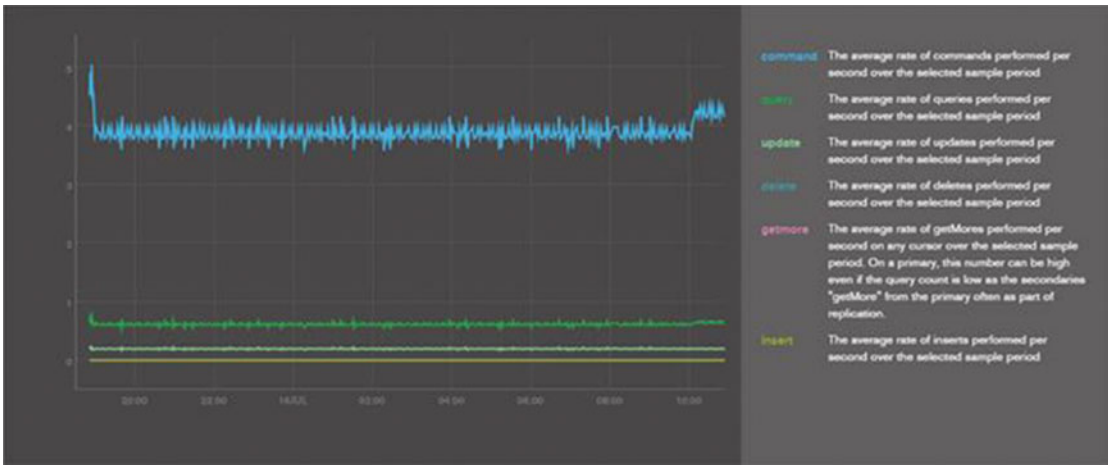


Figure 9-5. *Query response time*

Cloud Manager lets you do the following:

- Automate your MongoDB deployment (the configuration of MongoDB nodes, clusters, and upgrading of the existing deployment)
- Protect your data with continuous backup
- Provide any topology with AWS integration
- Monitor the performance in your dashboard
- Perform operational tasks such as adding capacity

For AWS users, it offers direct integration so the MongoDB can be launched on AWS without ever leaving Cloud Manager. You saw how to provision with AWS in Chapter tk.

Cloud Manager also helps you discover inefficiencies in your system and make corrections for smooth operation.

It collects and reports metrics using the agent you install. Cloud Manager provides a quick glance of the MongoDB system health and helps you identify the root causes of performance issues.

Next, you will look at the key metrics that should be used for any performance investigation. Along the way, you will also look at what the combination of the metric indicates.

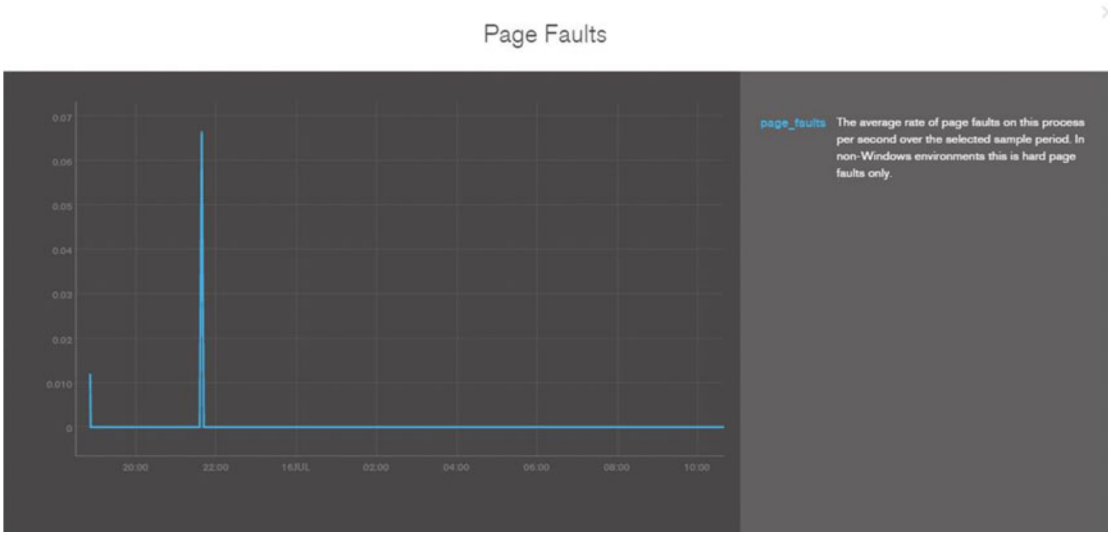
Metrics

You will be primarily focusing on the following key metrics; these metrics play a key role when investigating a performance problem issue. They provide an immediate glance of what's happening inside the MongoDB system and which of the system resources (i.e. CPU, RAM, or disk) are the bottlenecks.

- Page Fault
- Opcounters
- Lock percent
- Queues
- CPU time (IOWait and Users)

To view the below mentioned chart, you can click the Deployment link under Deployment Section. Select the MongoDB instance that has been configured to be monitored by Cloud Manager. Next, select required graphs/charts from the Manage Charts section.

Page fault shows the average number of page faults per second happening in the system. Figure 9-6 shows the page faults graph.



The average rate of page faults on this process per second over the selected sample period. In non-Windows environments this is hard page faults only.

Figure 9-6. *Page faults*

OpCounters shows average number of operations per second being performed on the system. See Figure 9-7.

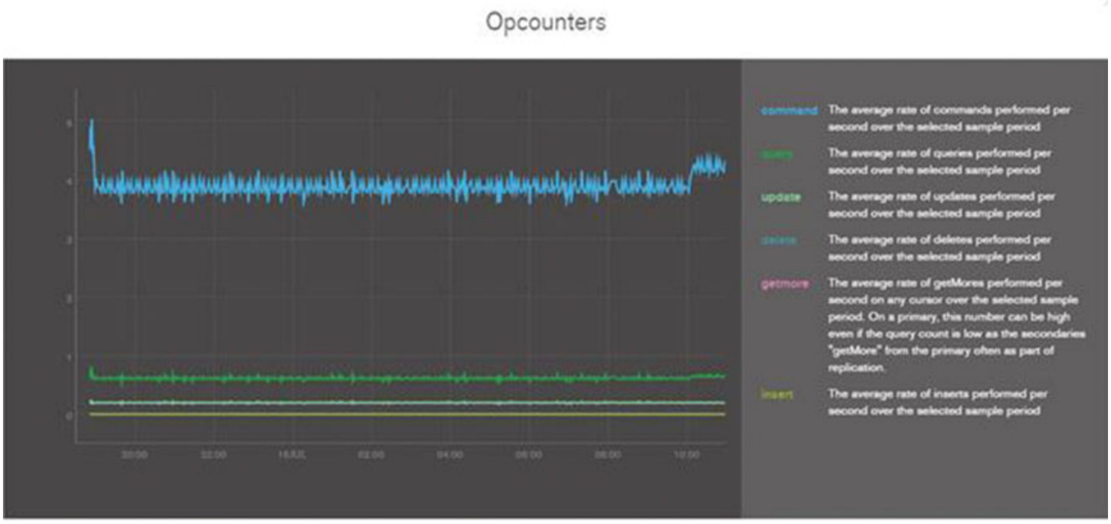


Figure 9-7. *OpCounters*

In the Page Fault to Opcounters ratio, the page faults depend on the operations being performed on the system and what's currently in memory. Hence a ratio of page faults per second to that of opcounters per second can provide a fair picture of the disk I/O requirement. See Figure 9-8.

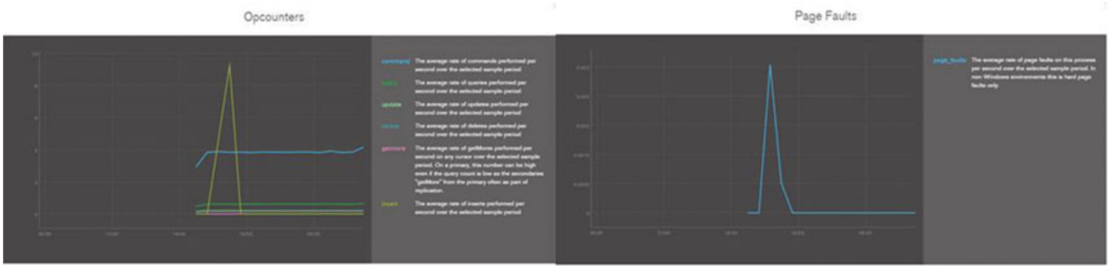


Figure 9-8. *Page fault to Opcounters ratio*

If the ratio is

- < 1, this classifies as low disk I/O.
- Near 1, this classifies as regular disk I/O.
- > 1, this classifies as high disk I/O.

The Queues graph displays the operations count waiting for a lock to be released at any given time. See Figure 9-9.

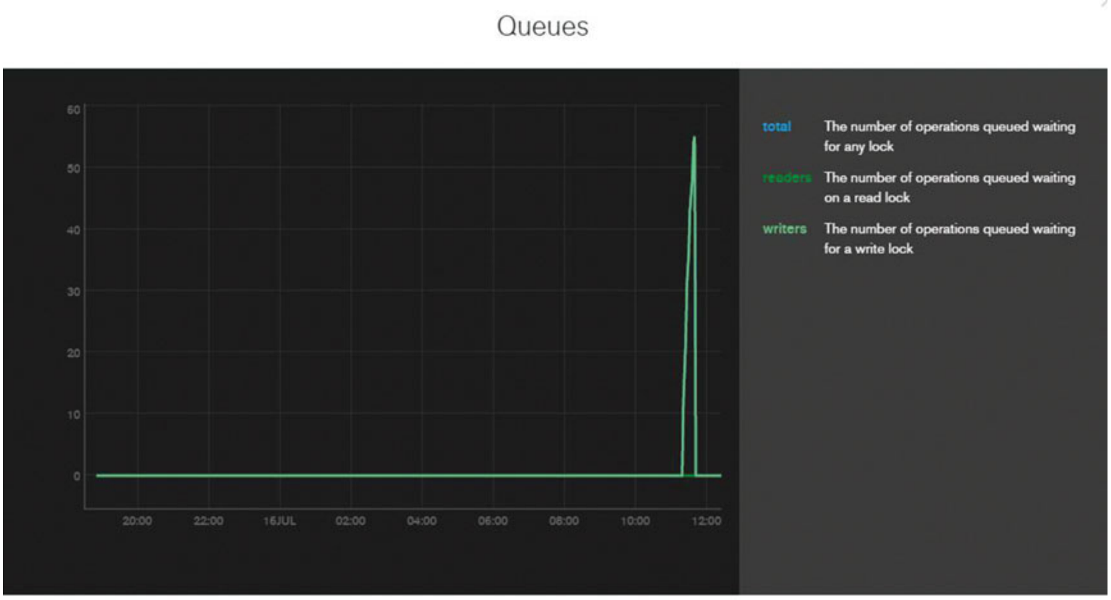


Figure 9-9. *Queues*

The CPU Time (IOWaits and User) graph shows how the CPU cores are spending their cycles. See Figure 9-10.

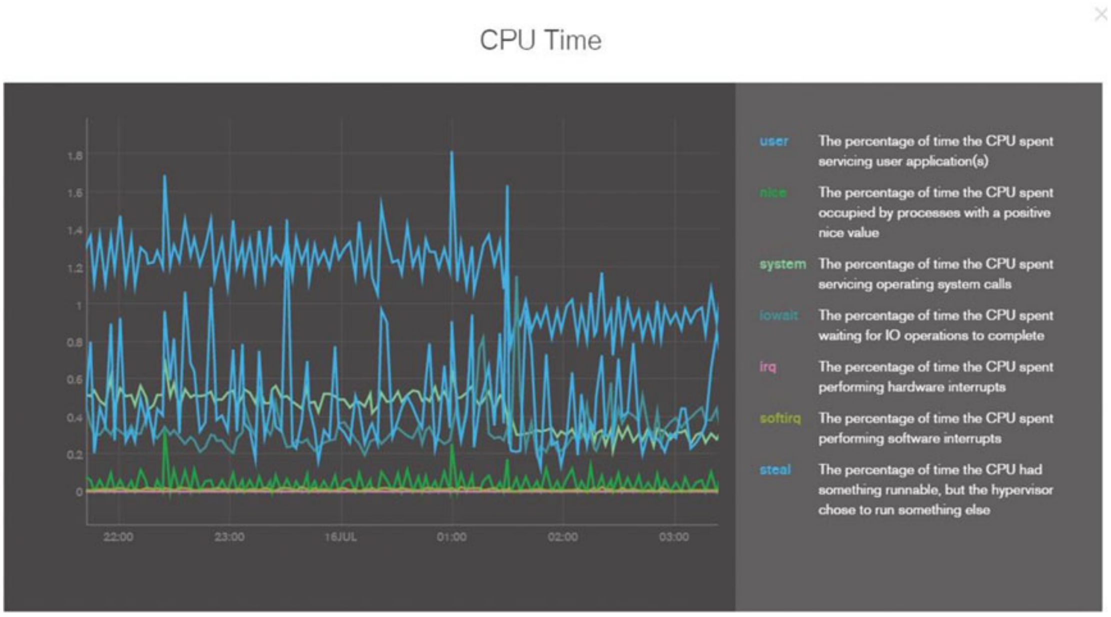


Figure 9-10. *CPU Time*

IOWait indicates the time the CPU spends waiting for the other resources, such as disks or the network. See Figure 9-11.

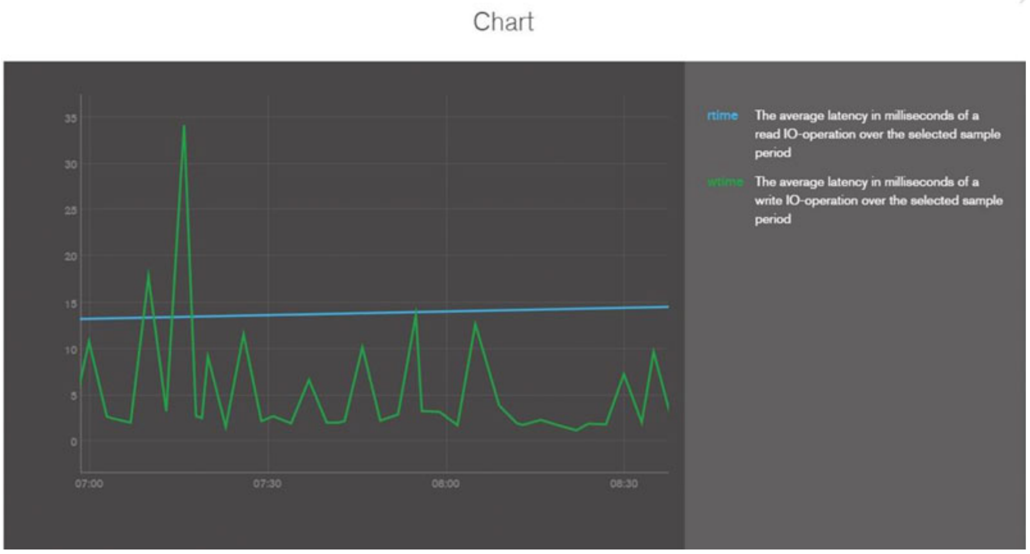


Figure 9-11. *IOWait*

User time indicates the time spent performing computations such as documents updating, updating and rebalancing indexes, selecting or ordering query results, or running aggregation framework commands, Map/Reduce, or server-side JavaScripts. See Figure 9-12.

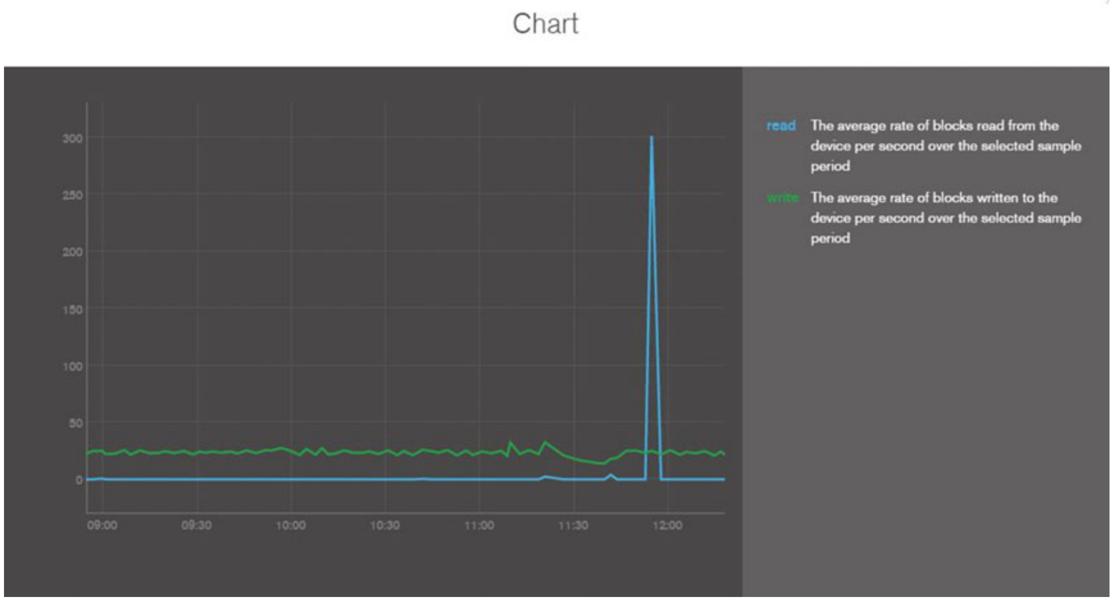


Figure 9-12. *User time*

To view the CPU Time graphs you need to install munin.

These key metrics and their combinations should be used to investigate any performance problems.

Summary

In this chapter you looked at how various utilities that are packaged as part of the MongoDB distribution can be used to manage and maintain the system.

You learned about the main operations that as an administrator you must be aware of for a detailed understanding of the utilities. Please read through the references. In the next chapter, you will examine MongoDB's use cases and you will also look at the cases where MongoDB is not a good choice.